Методичка 4.3 - Идентификация функций

Методы распознавания функций

По перекрестным ссылкам
По эпилогу
Про прологу

Функция - это основная структурная единица многих языков программирования, поэтому анализ дизассемблированного кода обычно начинается с идентификации функций и аргументов, которые передаются в функции.

Функция может принимать один или более аргументов, а может и вовсе не иметь входных аргументов. Может возвращать результат своей работы, а может и не возвращать. Ключевое свойство функции, что она возвращает управление на место ее вызова, а точнее на следующую инструкцию после ее вызова.

Вы уже немного знакомы с тем, как происходит вызов функции с помощью инструкции CALL, а также как происходит возврат значения с помощью инструкции RET.

Напомню, что инструкция CALL кладет в стек адрес следующей инструкции, а инструкция RET забирает этот адрес.

Таким образом, распознать функцию можно следующими признакам:

- По перекрестным ссылкам, которые строятся на основе инструкций CALL
- По эпилогу, который завершается инструкцией RET
- По наличию пролога функции

По этим признакам в большинстве случаев можно определить начало и конец функции.

Давайте попробуем на практике указанные методы.

Компиляций программы на языке Си

Специально для этой цели была подготовлена программа prog-3.3.

Исходный код программы prog-3.3

```
#include <stdio.h>

#pragma comment(lib,"user32.lib")

int sum_ascii(char *);
```

```
int main(int argc, char* argv[]) {

int result = 0;
char title[] = "Prog-3.3";

if (argc < 2) {
printf("\nUsage: prog-3.3.exe <license key>\n");
return 0;
}

result = sum_ascii(argv[1]);
if (result == 1000) {
printf("\nCongrats! Key is correct!\n");
} else {
printf("\nOops! Key is incorrect!\n");
}
return 0;
}

int sum_ascii(char *str) {
int sum = 0;
int len = strlen(str);
for(int i = 0; i < len; i++) {
sum += str[i];
}
return sum;
}
```

Компиляция:

```
> cl prog-3.3.c -Od /Gs- /GS- /link /DYNAMICBASE:NO /NXCOMPAT:NO
```

Давайте посмотрим на функции, которые получились после компиляции с помощью отладчика OllyDbg.

Идентификация функций

Первый признак - это перекрестные ссылки.

Первым делом необходимо найти все инструкции CALL. Содержимое их операнда и будет адресом начала функции.

В нашем примере четыре инструкции CALL. 3 из 4 адреса повторяются. Получается у нас всего 2 уникальных адреса.

0x00401000 и 0x00401120.

Существуют и более изощренные вызовы функций, например, через вычисление адреса в процессе выполнения программы, но мы их рассматривать в рамках этого курса не будем.

Иногда компилятор не использует инструкцию CALL для вызова функции, а использует, например, инструкцию JMP. Делается это для того, чтобы после выполнения функции была возможность управлять адресом возврата, например, для поддержки работы с исключениями. Например, если произойдет исключение, программа не должна продолжить свое выполнение в обычном режиме. В таком случае проанализировать все инструкции JMP уже не так просто, как инструкции CALL. Инструкций JMP в программе обычно сильно больше.

И здесь есть два пути. Либо не анализировать эти функции, либо учитывать при анализе, что начало новой функции должно быть за границей функции, из которой осуществляется JMP.

Второй признак - это пролог функции.

Почти все компиляторы при выключенной оптимизации помещают в начало функции следующие инструкции:

```
push ebp
mov ebp, esp
sub esp, <size>
```

Именно эти инструкции мы видим в начале функции main.

Зачем нужен пролог функции и как он работает мы разбирали на прошлом видео. Если в функции есть локальные переменные, то в прологе функции будет происходить корректировка указателя стека на размер области памяти, которая была выделена для локальных переменных.

В функции main происходит корректировка на 16 байт.

Третий признак - это эпилог функции.

Назначение эпилога функции разбиралось в прошлом видео.

```
mov   esp, ebp
pop   ebp
ret
```

Инструкция RET в эпилоге функции может служить уверенным признаком, что это место является завершением функции.

Давайте убедимся, что в других функция также присутствуют пролог и эпилог.

Переходим к первой функции - (Ctrl + G, 0x401000).

Переходим ко второй функции - (Ctrl + G, 0x401120).

Определять функции мы научились, теперь давайте рассмотрим, как можно определить аргументы функции.

Способы передачи аргументов в функцию

Существует всего 3 способа передачи аргументов в функцию: через стек, через регистры и комбинированный вариант.

Сами аргументы могут передаваться либо по значению, либо по ссылке. В первом случае передается копия значения, во втором указатель на переменную.

Существует несколько соглашений по передачи аргументов. Любое из них можно встретить при анализе программы:

- cdecl. Аргументы передаются через стек в порядке справа налево и стек очищает вызывающая функция.
- pascal. Аргументы передаются слева направо и стек очищает вызываемая функция.
- stdcall. Аргументы передаются справа налево, но стек очищает вызываемая функция.
- fastcall. Аргументы передаются через регистры.

При анализе функции первое, что стоит сделать - это определить какое соглашение используется в программе, чтобы понять, как функция принимает аргументы. Затем нужно определить количество этих аргументов и их тип.

Почти всегда соглашение можно определить по способу очистки стека. Если стек очищает вызывающая функция, то это cdecl, иначе, скорее всего, это stdcall.

Сложность может быть с определением порядка аргументов. Можно пойти на хитрость и определить порядок по библиотечным функциям, прототип которых нам известен. Если, конечно, программа их использует. Таким образом, по этим функциям определяем порядок передачи аргументов и, соответственно, в других функциях порядок будет таким же.

Определение количества аргументов тоже непростая задача. Если мы видим перед вызовом функции набор инструкций PUSH, это еще не означает, что все они используются для передачи аргументов. Возможно, это просто временное хранение значения из какого-нибудь регистра. Компилятор вполне может так сделать.

Легкого способа, который гарантированно позволяет определить количество входных аргументов в функцию не существует, но можно достаточно легко определить, сколько всего байт занимают все аргументы.

Можно посмотреть на коррекцию стека после инструкции CALL. Например, если после инструкции CALL идет инструкция ADD ESP, 8. Значит сумма байт всех аргументов равна 8.
На основе этой информации можно уже предположить какие инструкции PUSH перед вызовом функции относятся к передачи аргументов.

Определение типа является еще более сложной задачей. Передачу целых чисел и строк можно легко определить при динамической отладке. Как правило, строки передаются по указателю, значит по адресу. В процессе отладки можно посмотреть, что находится по адресу, который был передан и увидеть содержимое строки. С более сложными структурами данными так не всегда получается. Если строки будет передаваться по значению, то скорее всего вы увидите инструкцию MOVS. С числами всё еще проще, они передаются в явном виде.

Давайте попробуем определить входные аргументы для примере специально подготовленной программы prog-3.3.

Идентификация аргументов функции

Продолжим анализ программы prog-3-3 под отладчиком OllyDbg.

Сначала нам нужно определить, как передаются аргументы в нашей программе. Смотрим вызов функции, которая находится по адресу 0x00401000 и видим, что после вызова идет корректировка стека. Это значит, что корректировка выполняется в вызывающей функции. Получается, что используется соглашение cdecl.

Нам нужно найти функцию, которая гарантированно принимает на вход более одной переменной.

Мы знаем, что наша программа принимает входные аргументы. Вспоминаем прототип функции main (int argc, char* argv[]). Сначала передается число, затем массив аргументов.

Переходим в код перед вызовом функции main (Правой кнопкой на первой инструкции функции - Go to - Call from ...).

Видим, что перед вызовом функции main выполняется 3 раза инструкция PUSH. Это передача параметров в функцию main.

```
PUSH EDI
PUSH ESI
PUSH DWORD PTR DS:[EAX]
```

Через регистры ESI и EDI у нас передаются границы данных argv, а далее, видимо, значение argc. Давайте проверим.

Поставим точку на первой инструкции PUSH. Перезапускаем программу (Ctrl + F2). Остановились на инструкции PUSH ESI.

В окне Dump поставим адрес, который содержится в регистре ESI.

В данных видим полный путь до нашей программы, значит это argv. Через регистр EDI передается граница параметра argv. Выполняем эти инструкции и доходим до инструкции PUSH DWORD PTR DS:[EAX].

В окне Dump поставим адрес, который содержится в регистре EAX.

В данных видим число 1. Это и есть количество аргументов. По умолчанию всегда один аргумент, это название программы.

Давайте изменим количество аргументов и проверим, так это или нет.

Debug - Arguments. Указываем произвольный набор символов. Перезапускаем программу (Ctrl - F2).

Видим, что теперь там значение 2. Значит, это точно argc. Таким образом, мы выяснили, что аргументы передаются справа налево, значит это точно cdecl.

Теперь нам нужно определить суммарный размер входных аргументов. Видим, что после вызова функции main идет инструкция ADD ESP, 0C. Это значит, что суммарно было передано 12 байт. Действительно, так и есть, выше мы видим 3 инструкции PUSH по 4 байта.

Давайте посмотрим на на примере других функций. Переходим во внутрь функции main. После каждой инструкции CALL мы видим корректировку стека. После вызова функции по адресу 0x401000 идет инструкция ADD ESP, 4. И перед вызовом мы видим одну инструкцию PUSH, значит это передача аргумента в функцию.

Далее можно попробовать определить тип аргумента. Обычно аргументом является либо адрес, который на что-то указывает, либо число. Пример с числом мы видели при анализе аргумента argc для функции main.

Пример со строкой можно увидеть при вызове функции по адресу 0x004010000. Выполняется инструкция PUSH EDX. В этот момент в окне можно посмотреть, что там находится (Ctrl + G, EDX). В данных мы видим аргумент, который мы передали при запуске программы.